# WinFS Replication

Doug Terry
Microsoft Research Silicon Valley

June 2003

# WinFS Synchronization & Caching Team

***Mission:*** *Provide a state-of–the-art synchronization platform integrated into and leveraging WinFS in support of diverse applications requiring peer-to-peer replication for data sharing, high-availability, and offline access.*

Irena Hudis – Product Unit Manager
Lev Novik – Architect



# Talk Outline

This morning:

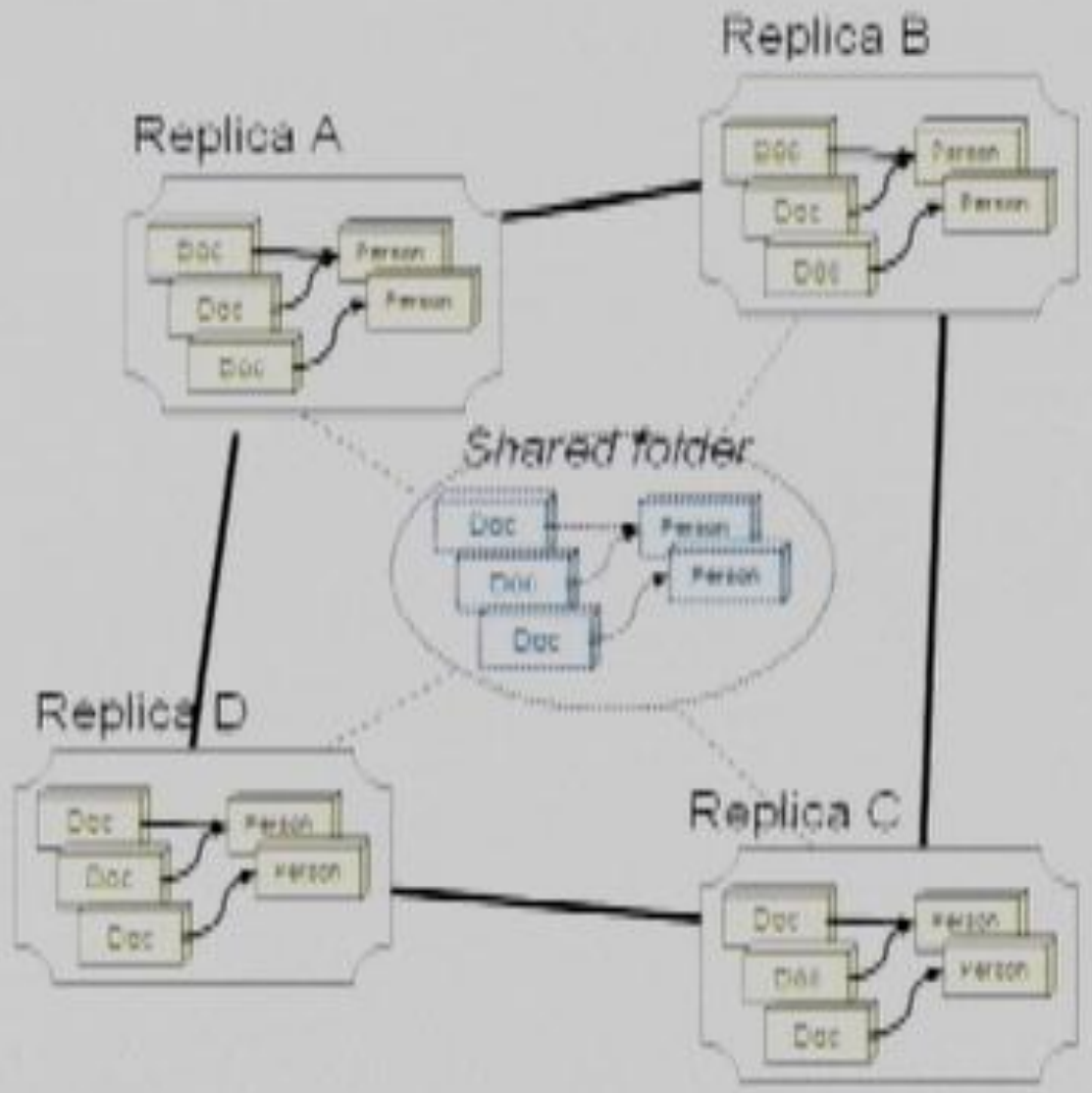
- Goals and requirements
- Change units, versions, knowledge
- Update propagation
- Conflict detection and resolution

This afternoon:

- Filtered sync
- Meta-data cleanup
- Open Issues



# Goal: Replication of WinFS Folders



# WinFS Synchronization Requirements

- Allow multi-master updates
- Support peer-to-peer topologies
- Operate at the level of WinFS Schema
- Permit configuration of "what" and "when" to sync
- Detect and handle conflicts during synchronization
- Accommodate varying network characteristics: intermittent connections, slow networks, offline
- Provide out-of-the-box "zero admin" manageability, resilience, and ease of use
- Guarantee convergence, even in "unmanaged" environments



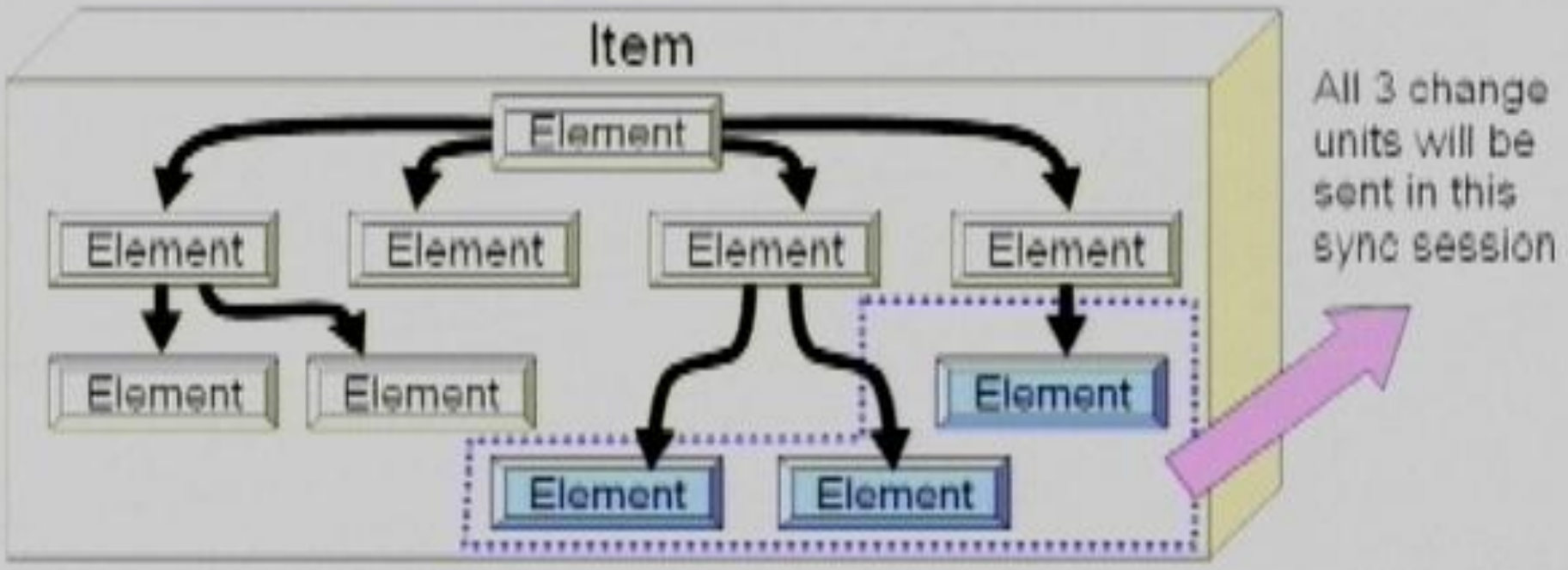
# WinFS Synchronization – Approach

- Read-anywhere/update-anywhere model
- Pair-wise reconciliation between replicas
- User-defined topology and schedule
- "Net Changes" of WinFS items/folders are conveyed, not individual actions
- Adapters for replication with non-WinFS machines, services, and devices (*not discussed in this talk*)



# Items vs. Change Units

- Item = granularity of consistency
- Change unit = granularity of update, propagation, and conflict detection
  - elements or fields of elements



# Versions of Changes

- Each change has a globally-unique id (version)
- Version = ID of the replica making the change + replica-specific number
  - Replica IDs are GUIDs
  - Replica-specific number is ever-increasing at the replica
  - e.g. A5, B2, C7
- Version of change unit is version of last change applied to that element
  - e.g. I.c: A5
- Versions only used by synchronization protocol; not visible to WinFS users



# Knowledge for Update Propagation

- Knowledge = concise description of the set of changes received by a replica
  - knowledge depends only on local database
  - not pair-specific: not "what I have received from you"
- When requesting changes, specify
  - Current knowledge (i.e. *"don't send these"*)
  - Filter (i.e. *"only interested in these"*)
- When enumerating changes
  - Return change units and tombstones not covered by requester's knowledge
- When conveying changes, specify
  - Version of the change
  - The change itself
  - "Made with" knowledge – for conflict detection
  - "Learned" knowledge – not to send them again



# Knowledge Abstractly

Main operations on knowledge:

- Test if a given knowledge covers a given change
  - "Does he already know about this change?"
- Add a change version to one's knowledge
  - "I just learned of a new change."
- Merge one knowledge with another to produce combined knowledge
  - "I knew this and he knew that, and now we both know this."



# Knowledge Implementation

- Version vector = set of (replica GUID, replica-specific number) pairs
  - e.g. X4 Y3 Z7
- Semantics: "all versions authored by this replica up to this number"
  - e.g. shorthand for: X1 X2 X3 X4 Y1 Y2 Y3 Z1 Z2 ... Z7
- Exceptions needed for changes received out-of-order
  - e.g. X4 Y3 Z7 +X6
- Includes every replica that ever made a change



# Protocol Using Knowledge

Machine A

i: A4
j: B3
k: C5
l: A2

Knowledge:

A4 B3 C6 D0

Machine B

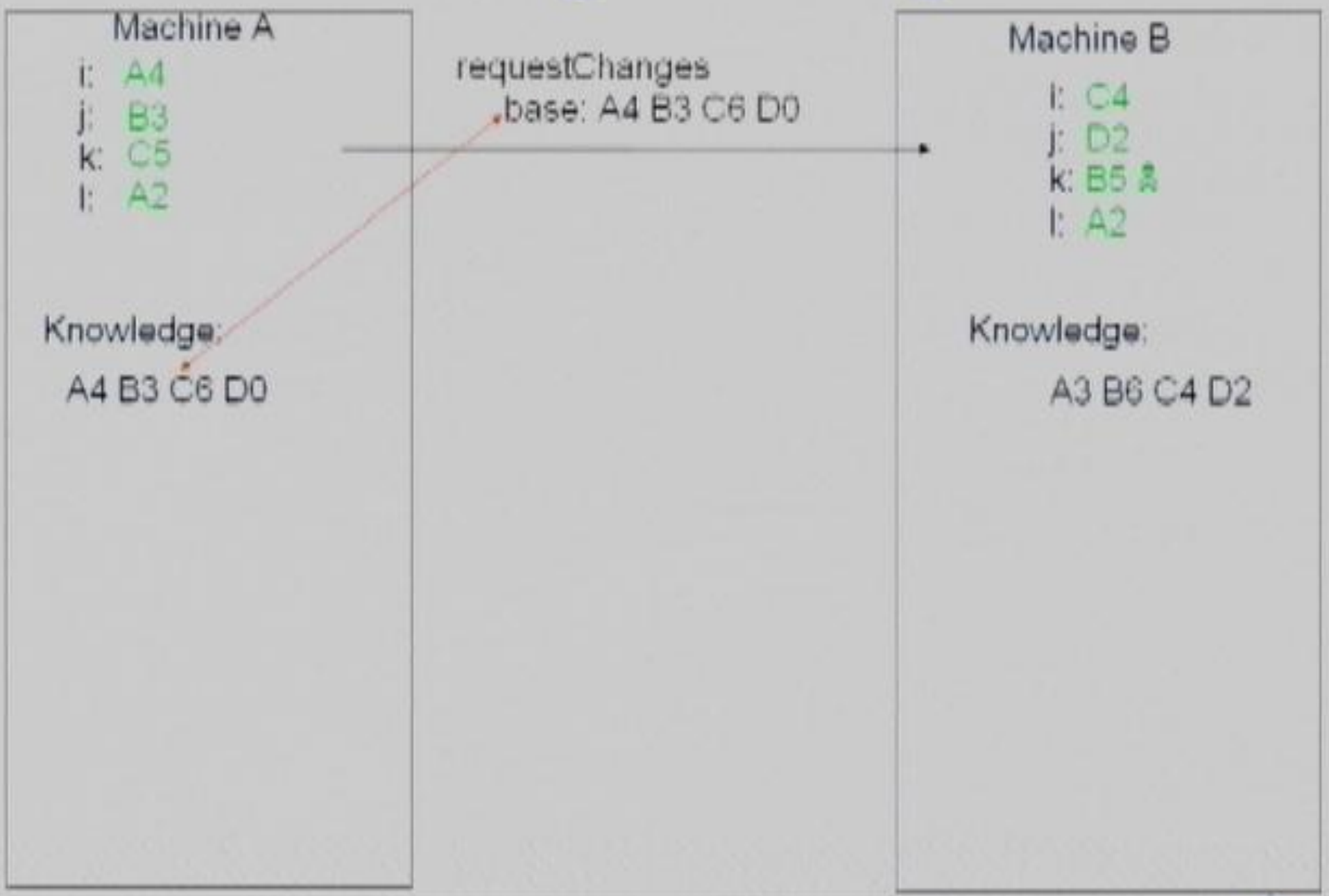
l: C4
j: D2
k: B5 &
l: A2

Knowledge:

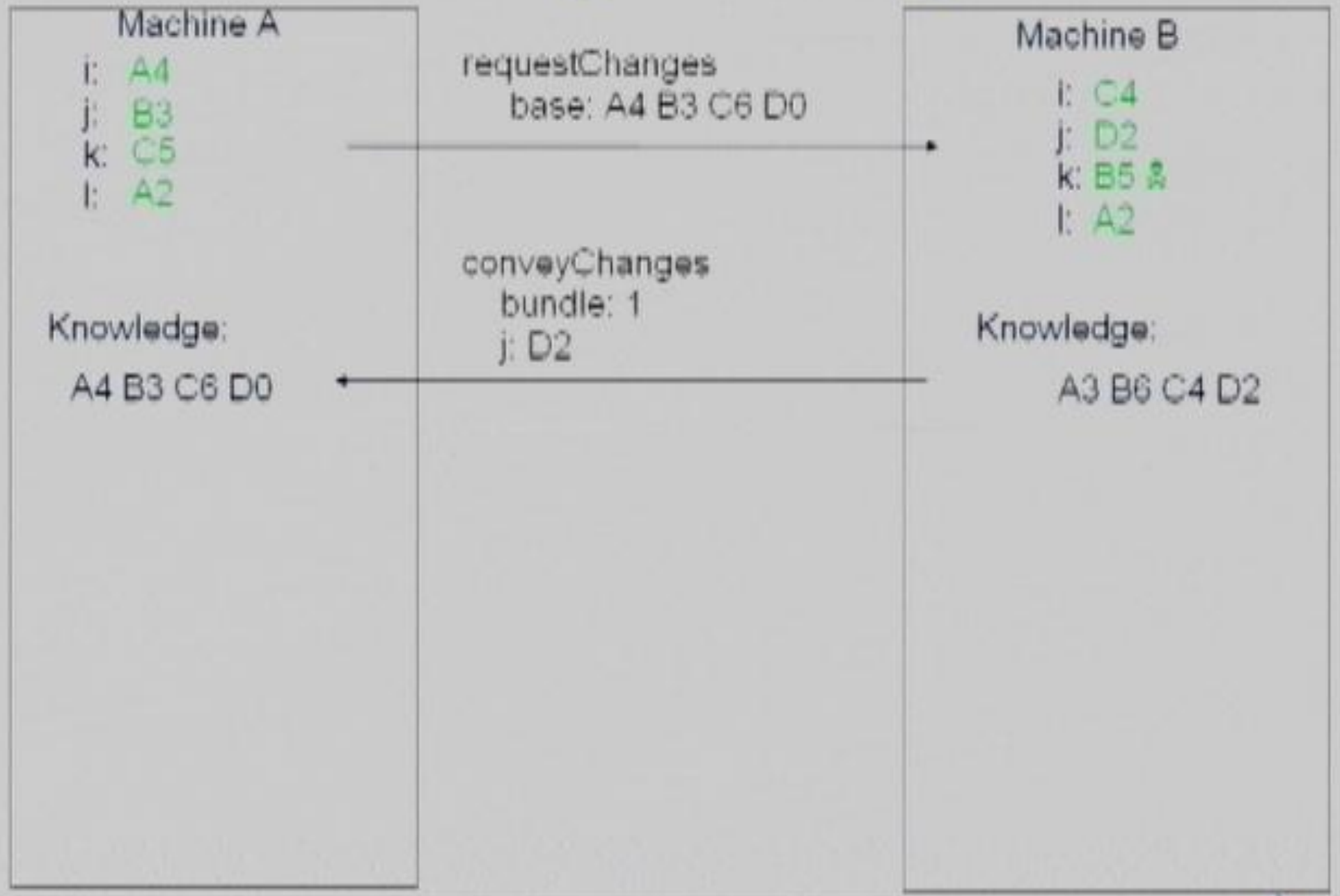
A3 B6 C4 D2



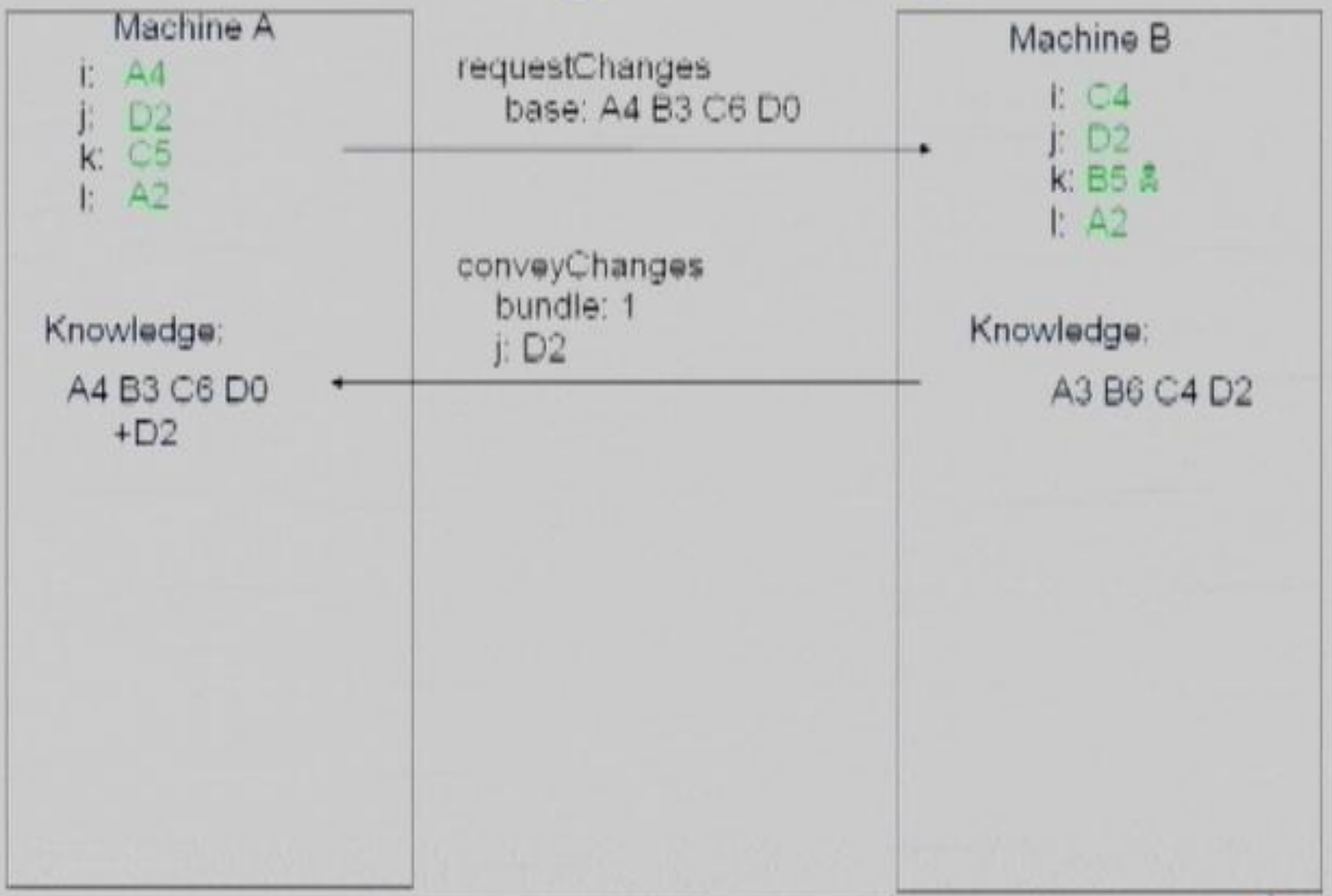
# Protocol Using Knowledge



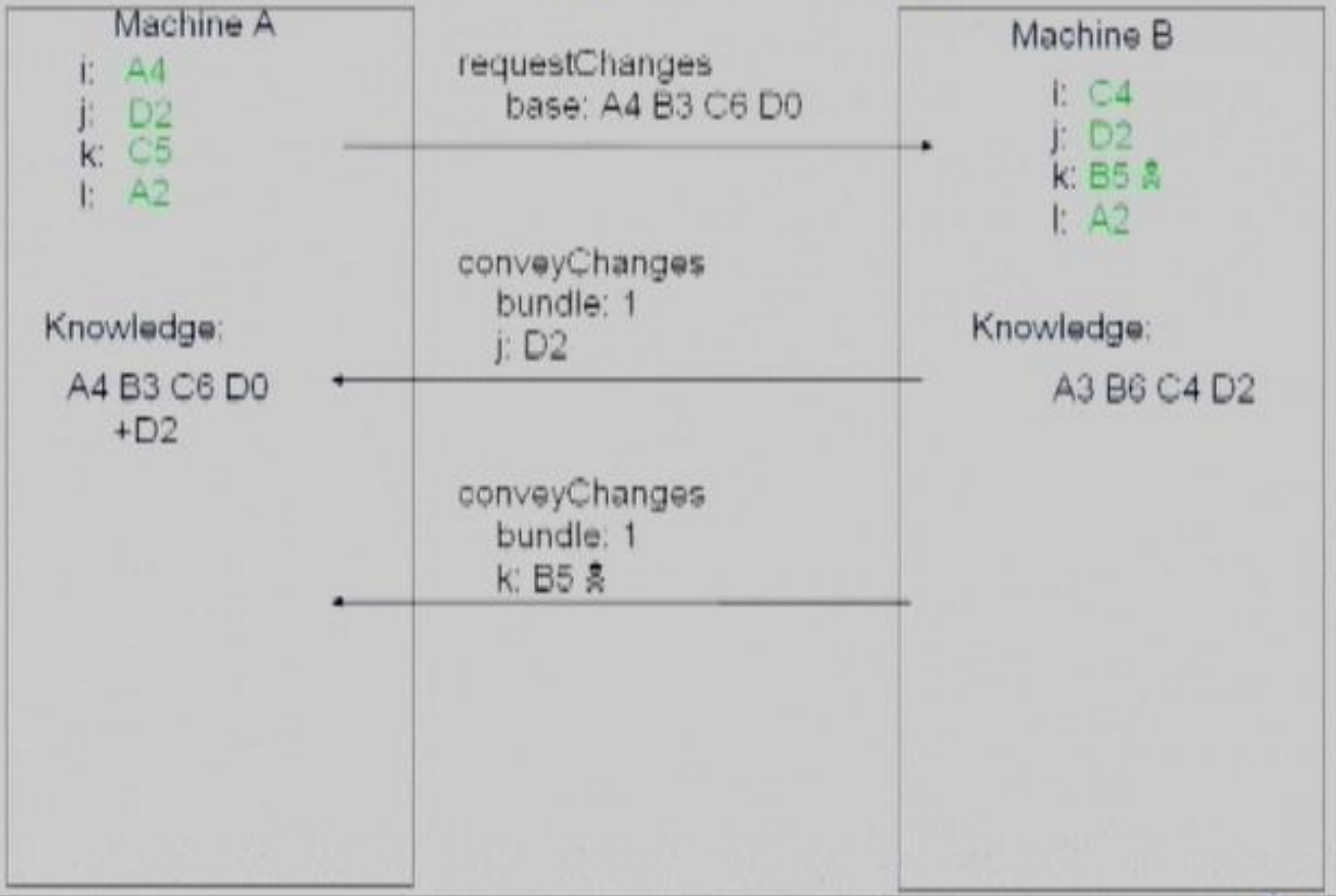
# Protocol Using Knowledge



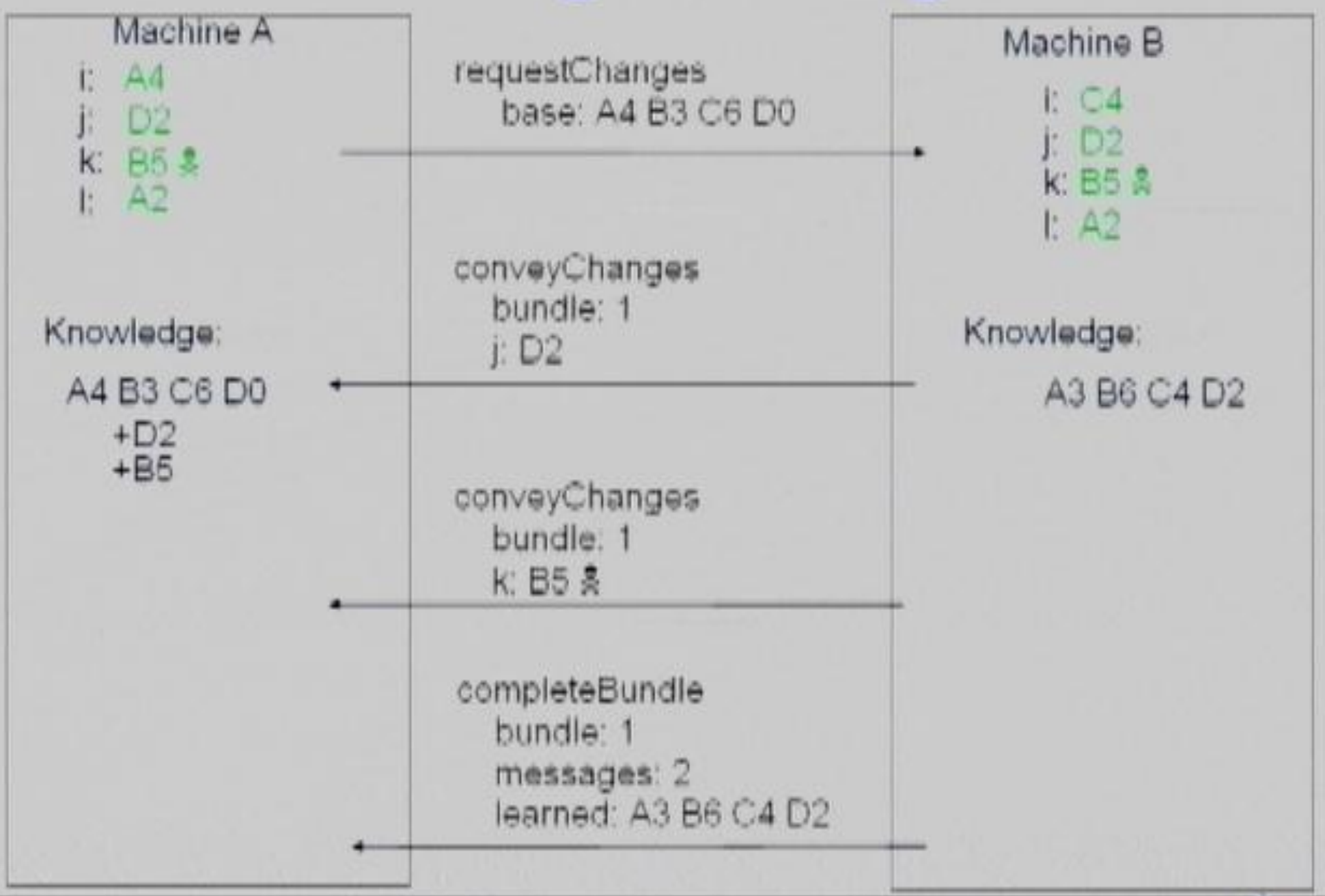
# Protocol Using Knowledge



# Protocol Using Knowledge



# Protocol Using Knowledge



# Sync Protocol : Things Not Shown

- Conflict detection and resolution
  - Detecting concurrent updates
  - Logging or automatically resolving conflicts
- Batching
  - Changes are sent in batches to reduce noise/content ratio
  - Changes ordered to optimize resumption after connection loss
- Topology optimizations
  - Read-only and slave replicas do not figure in knowledge vectors
- Filtering
  - Synchronizing only certain items as well as only certain portions of items is supported
- Meta-data cleanup
  - Removing tombstones
  - Pruning inactive replicas



# Conflict Definition

- Two changes conflict iff

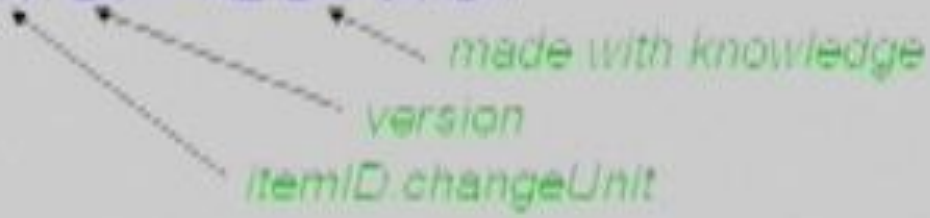
(1) the semantics of the changes are incompatible
  - Overlap-based (affecting the same change unit)
  - Constraint-based (applying both would violate a constraint)

(2) the changes were performed concurrently
(without knowledge of each other)



# Conflict Detection using Knowledge

- "Made with" knowledge = knowledge of replica when change was accepted
  - Stored and propagated with each change (conceptually)
  - e.g. I.c: B5 – A2 B4 C6 D1




- Changes $C_1$ and $C_2$ are concurrent iff
  (1) $C_1$.version not in $C_2$.madeWithKnowledge and
  (2) $C_2$.version not in $C_1$.madeWithKnowledge



# Protocol with Conflict Handling

Machine A

i: A4 – A3 B2 C4 D0
j: B3 – A1 B2 C2 D0
k: C5 – A2 B3 C4 D0
l: A2 – A1 B1 C1 D0

Knowledge:

A4 B3 C6 D0

Conflict Log:

Machine B

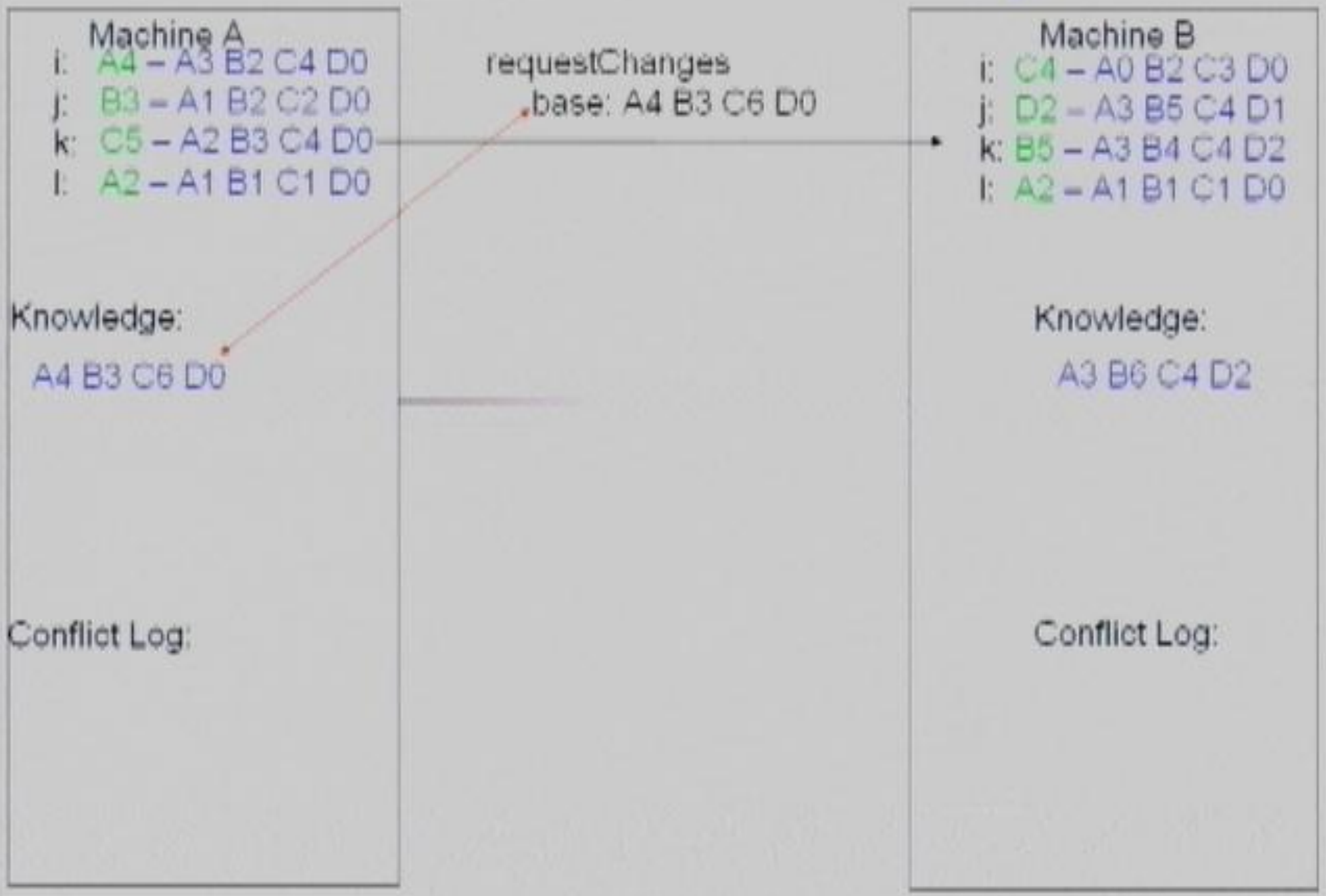
i: C4 – A0 B2 C3 D0
j: D2 – A3 B5 C4 D1
k: B5 – A3 B4 C4 D2
l: A2 – A1 B1 C1 D0

Knowledge:

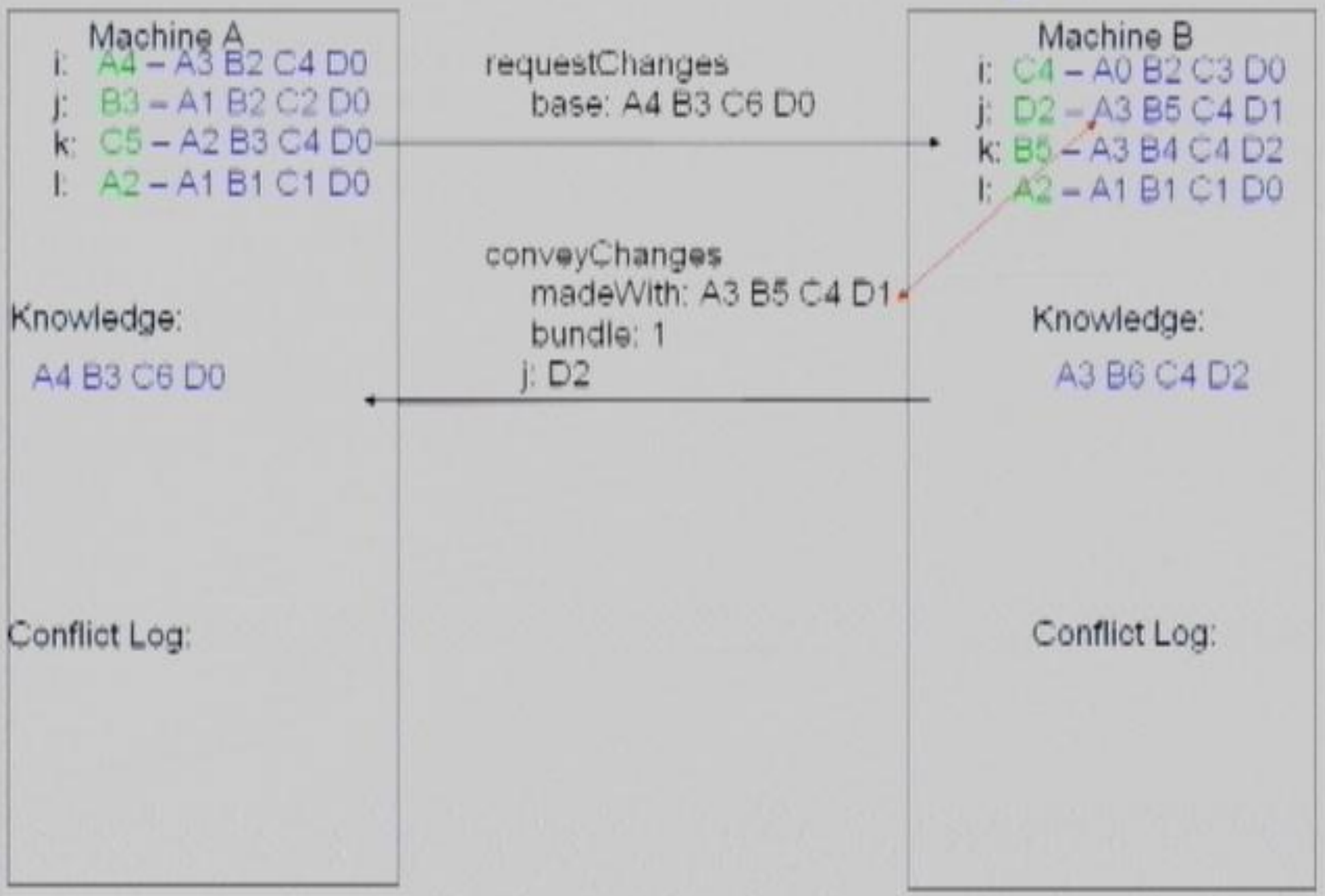
A3 B6 C4 D2

Conflict Log:

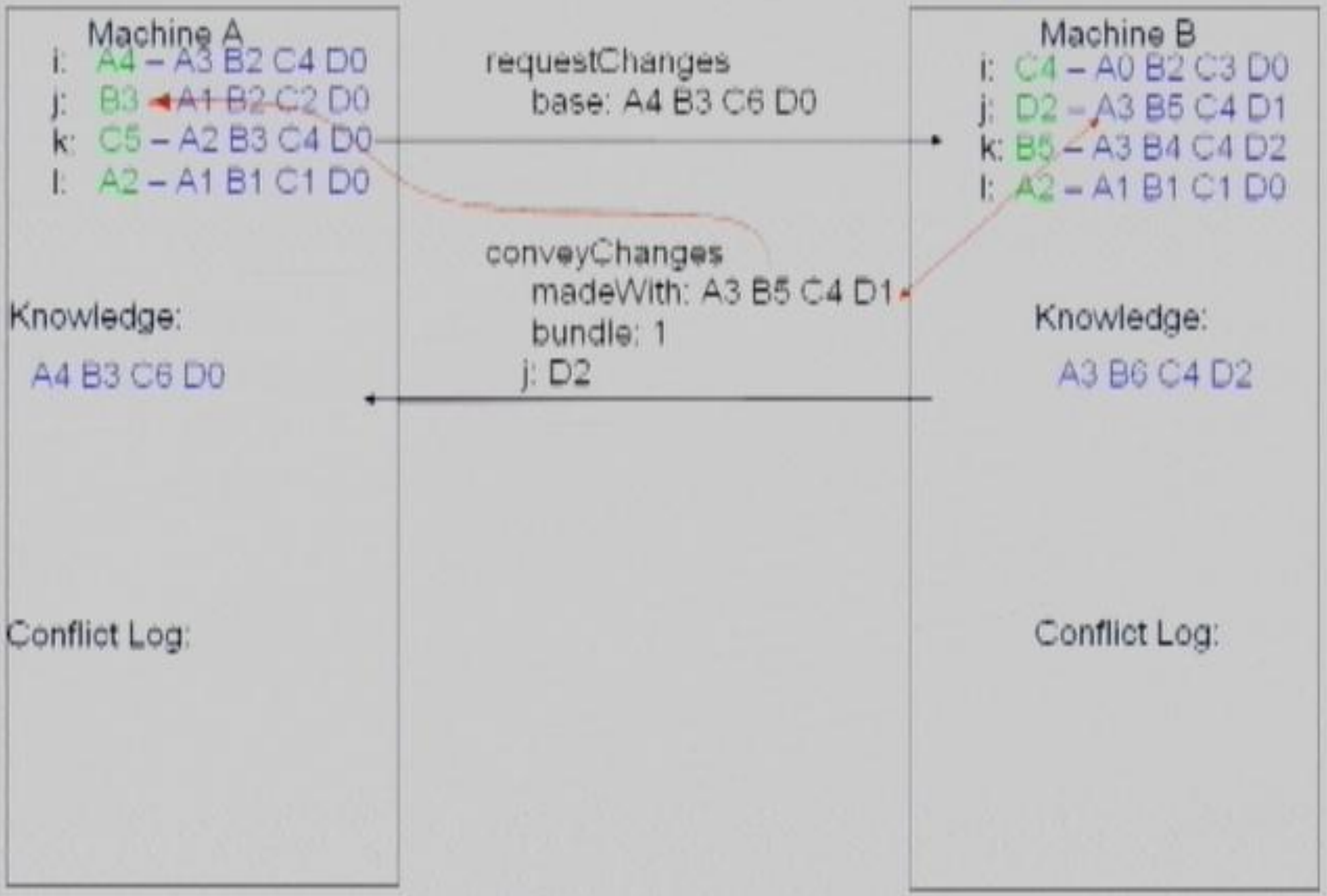
# Protocol with Conflict Handling



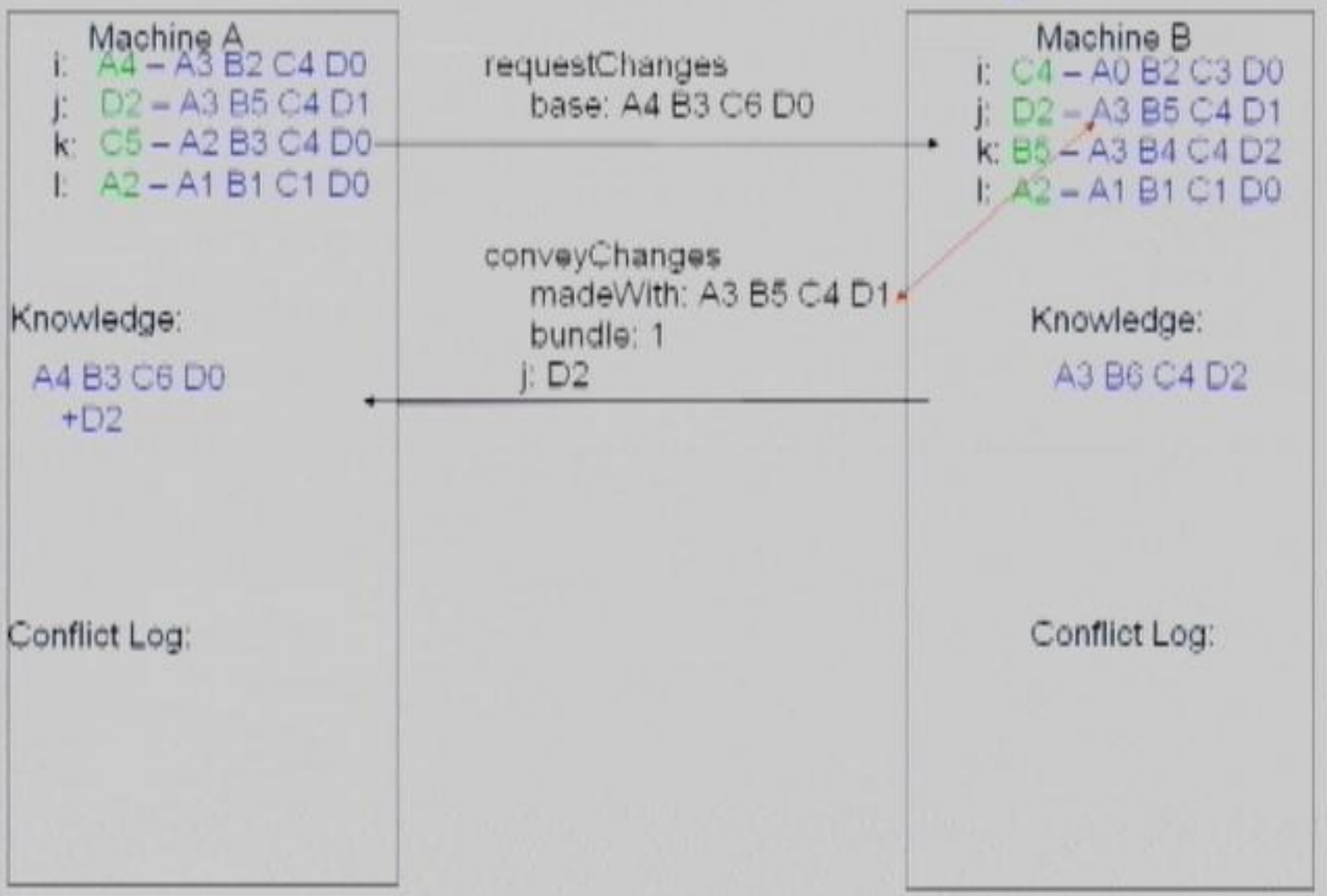
# Protocol with Conflict Handling



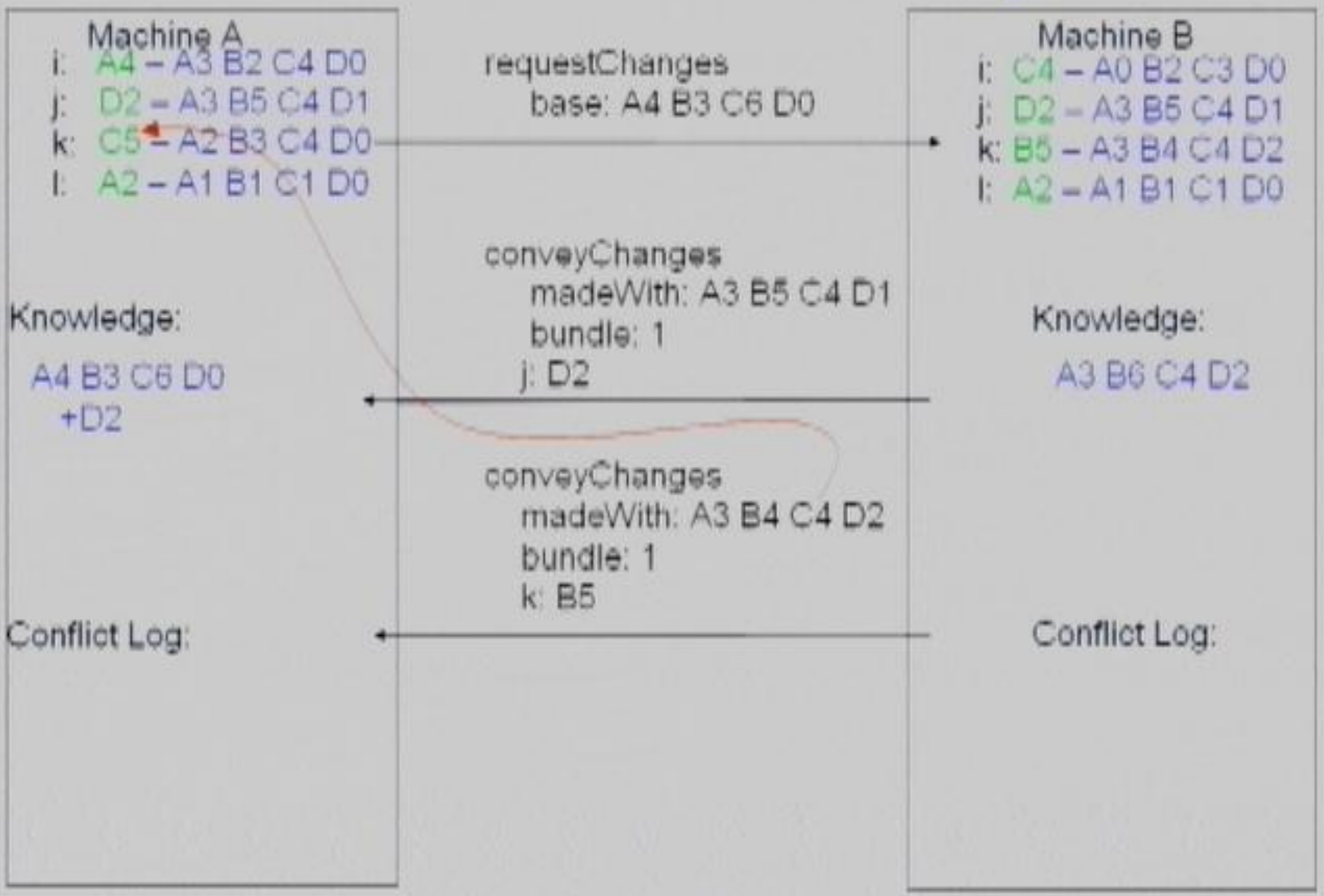
# Protocol with Conflict Handling



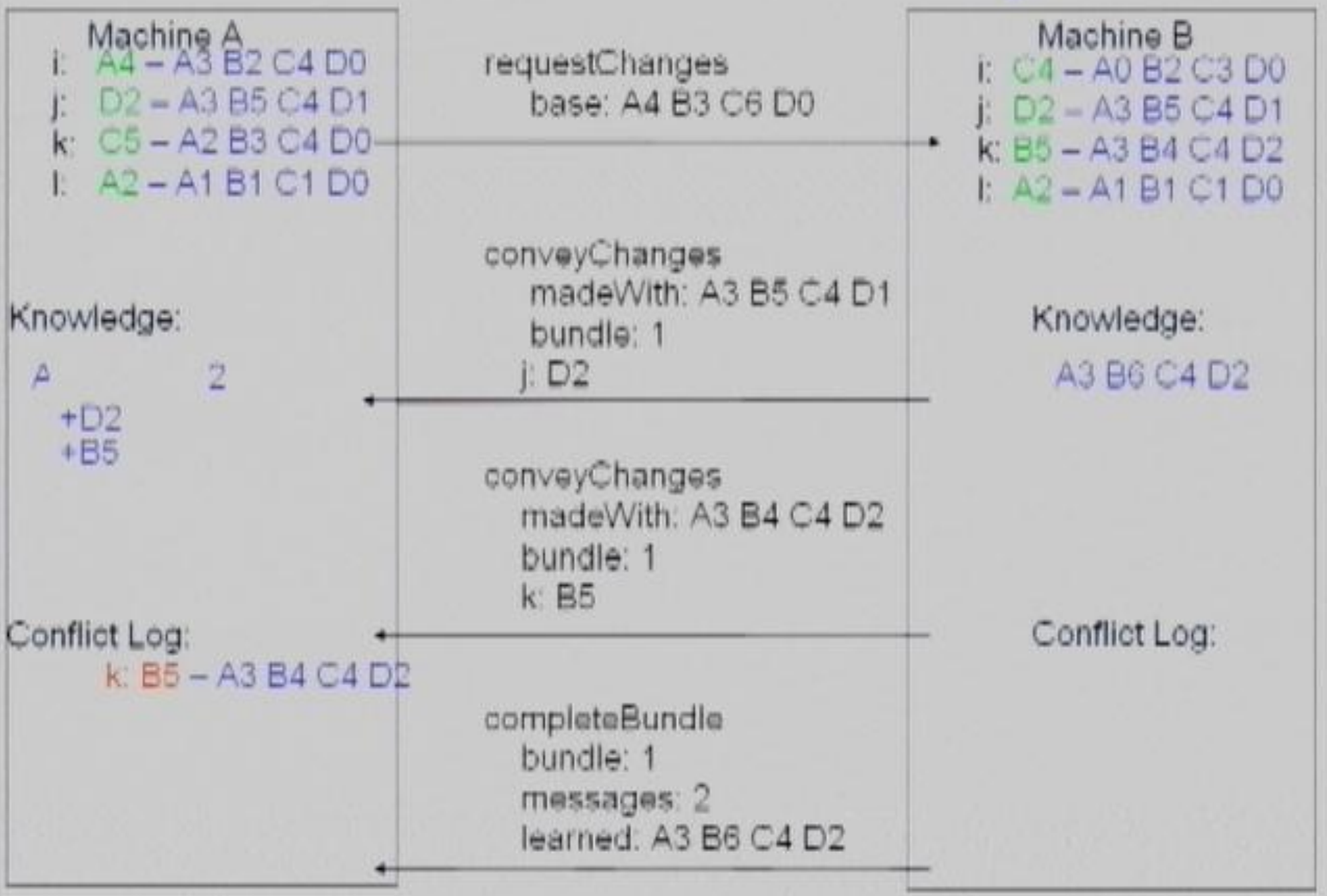
# Protocol with Conflict Handling



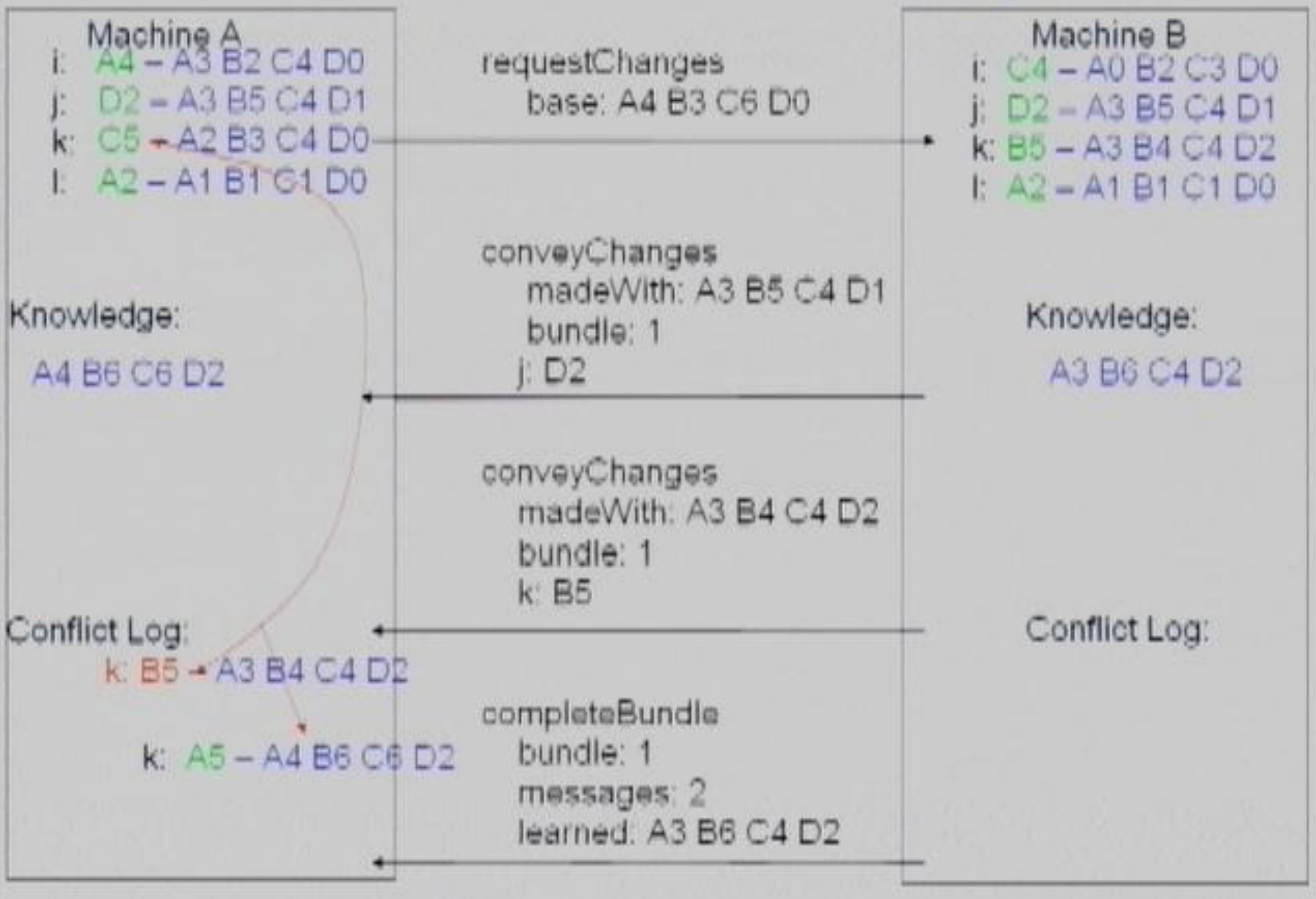
# Protocol with Conflict Handling



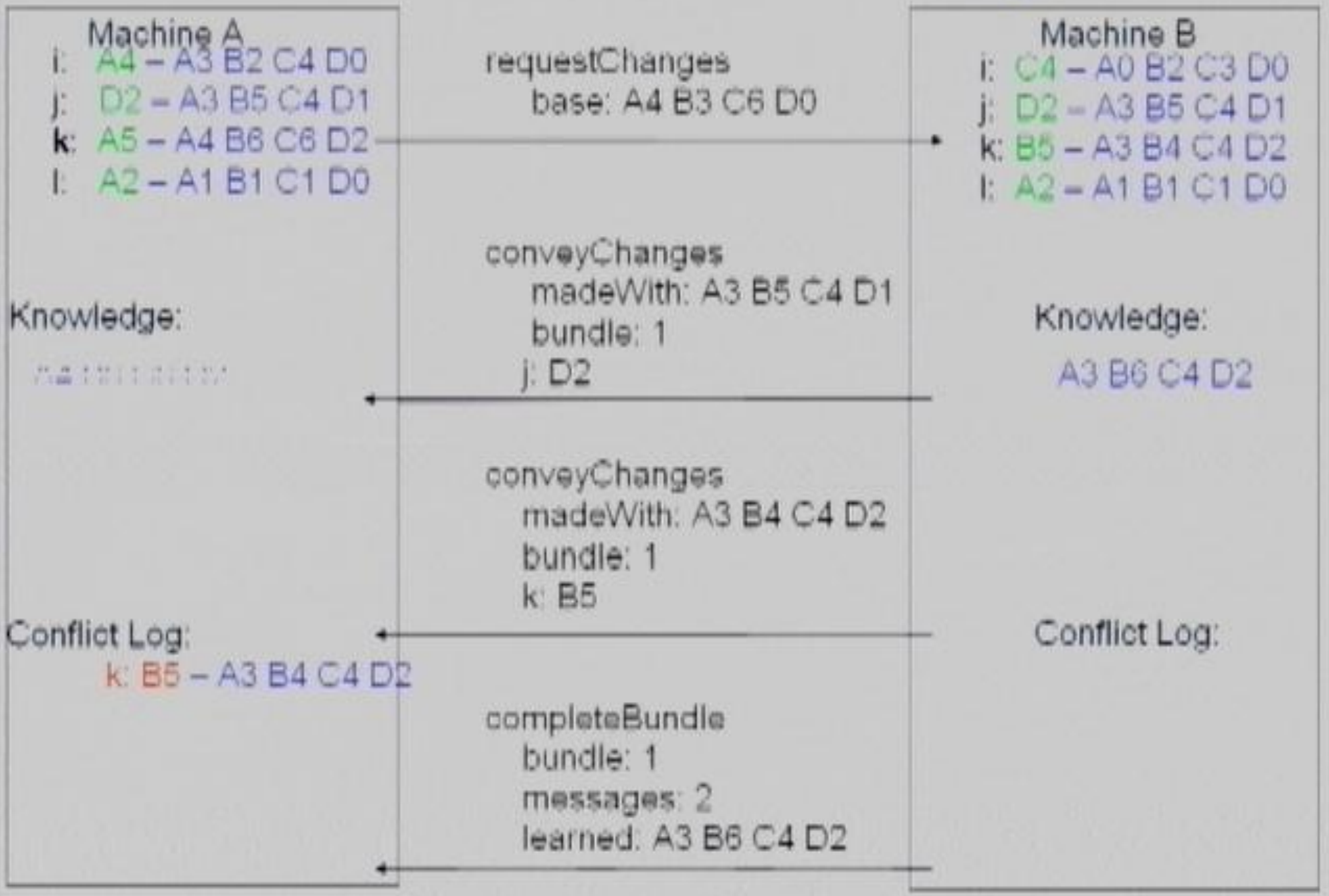
# Protocol with Conflict Handling



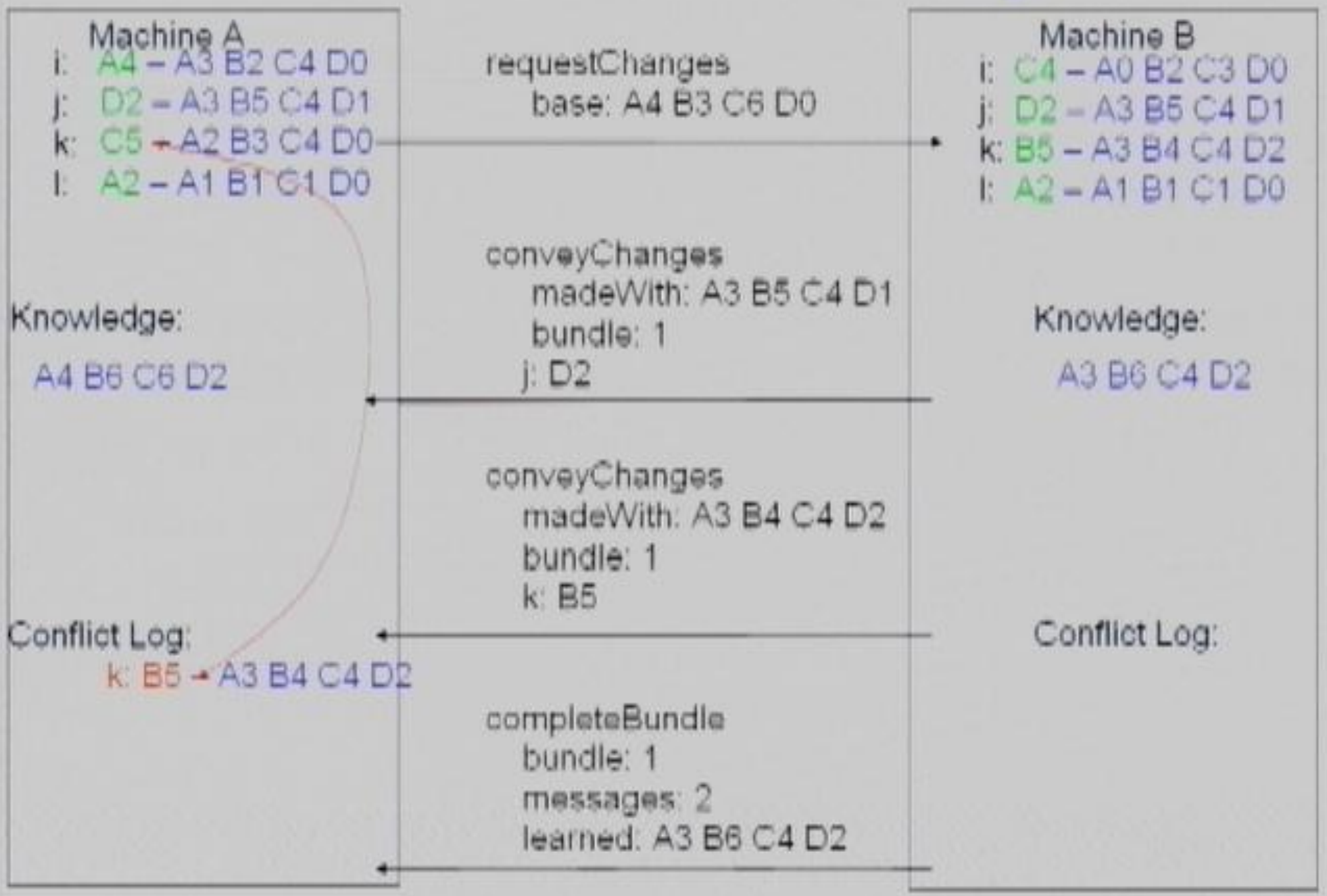
# Protocol with Conflict Handling



# Protocol with Conflict Handling



# Protocol with Conflict Handling

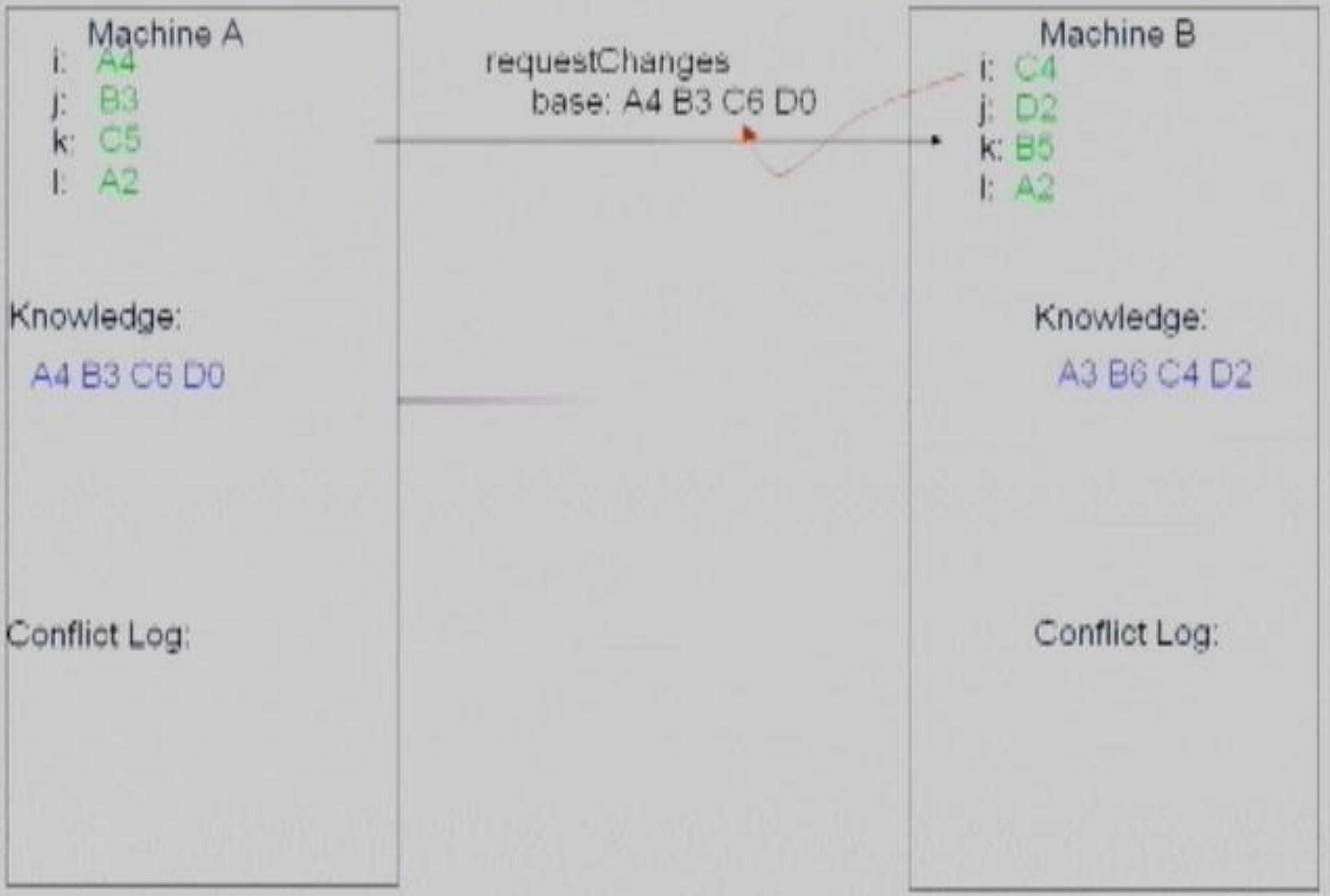


# Protocol with Conflicts: Not Shown

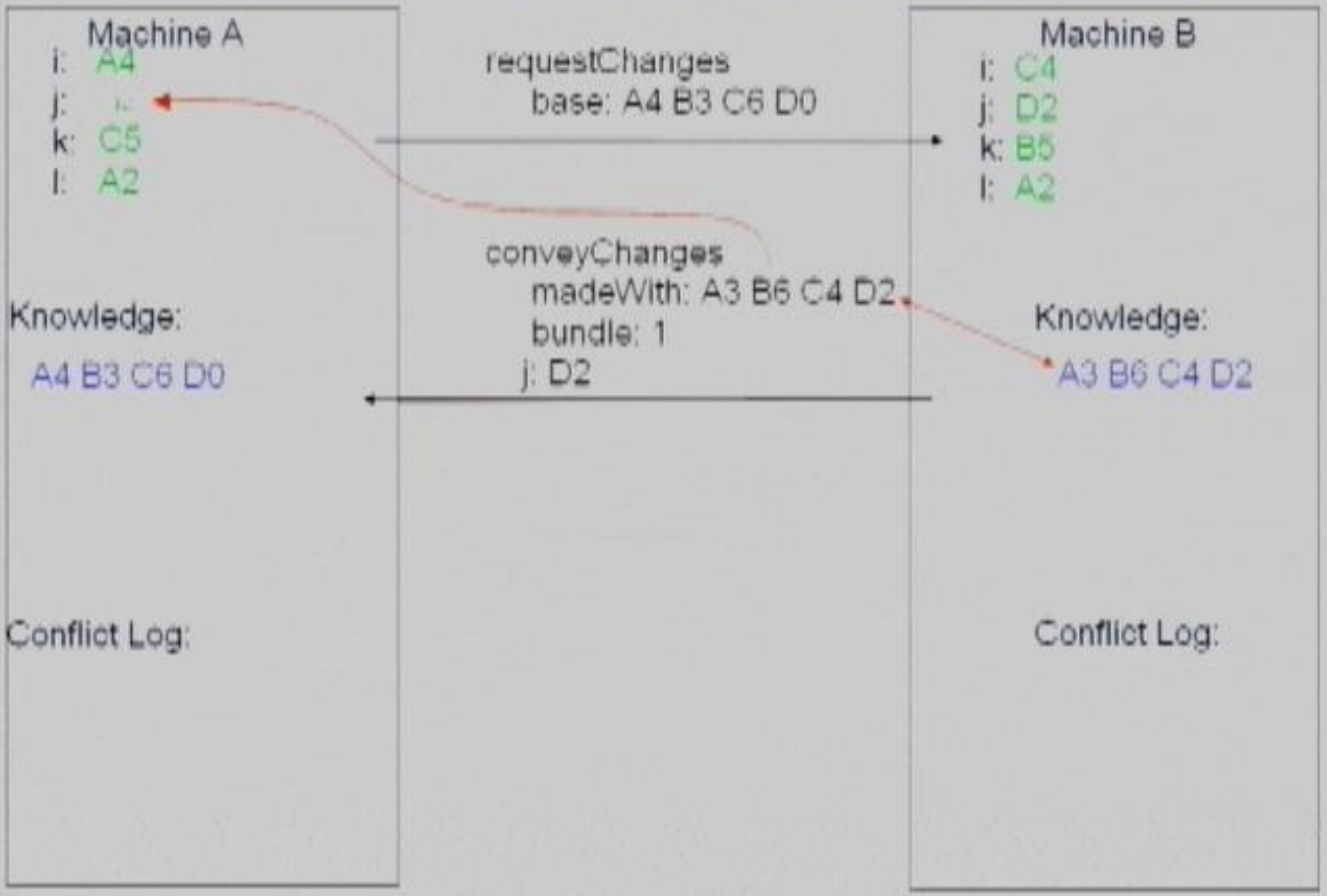
- Automatic Conflict Resolution
  - Occurs synchronously, immediately after conflict detection
  - Conflict Handler selected according to policy and conflict info
  - Conflict Handlers choose one of the following *outcomes*: local wins, remote wins, propose new, reject
  - Conflict resolutions propagate to other replicas
    - May cause additional conflicts
- Storage Optimization
  - Do not need to store made with knowledge "unless" an item is in conflict!
  - Use current base knowledge as made with knowledge
    - Intuition: item could have been updated with the current knowledge



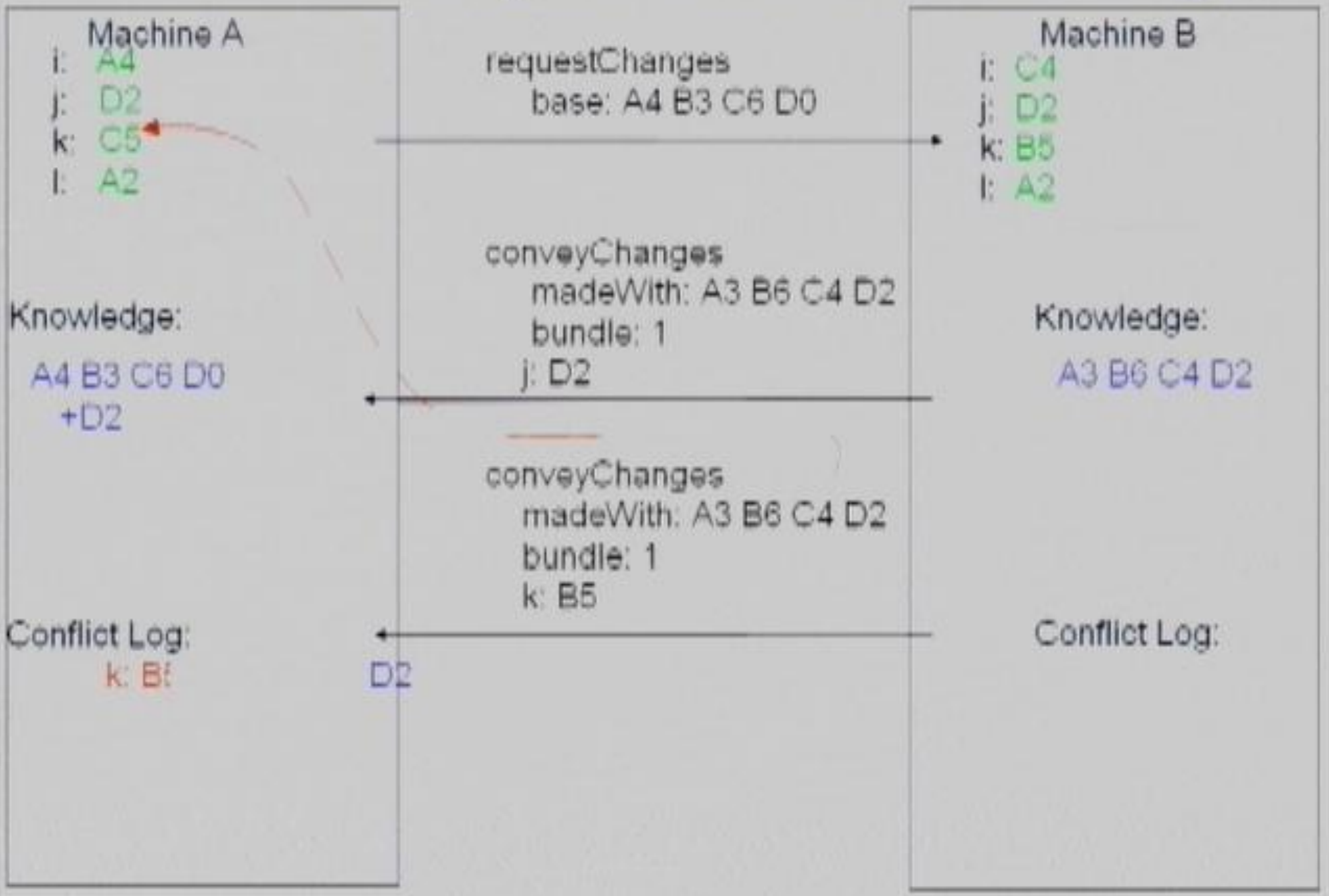
# Protocol with Implicit "Made With" Knowledge



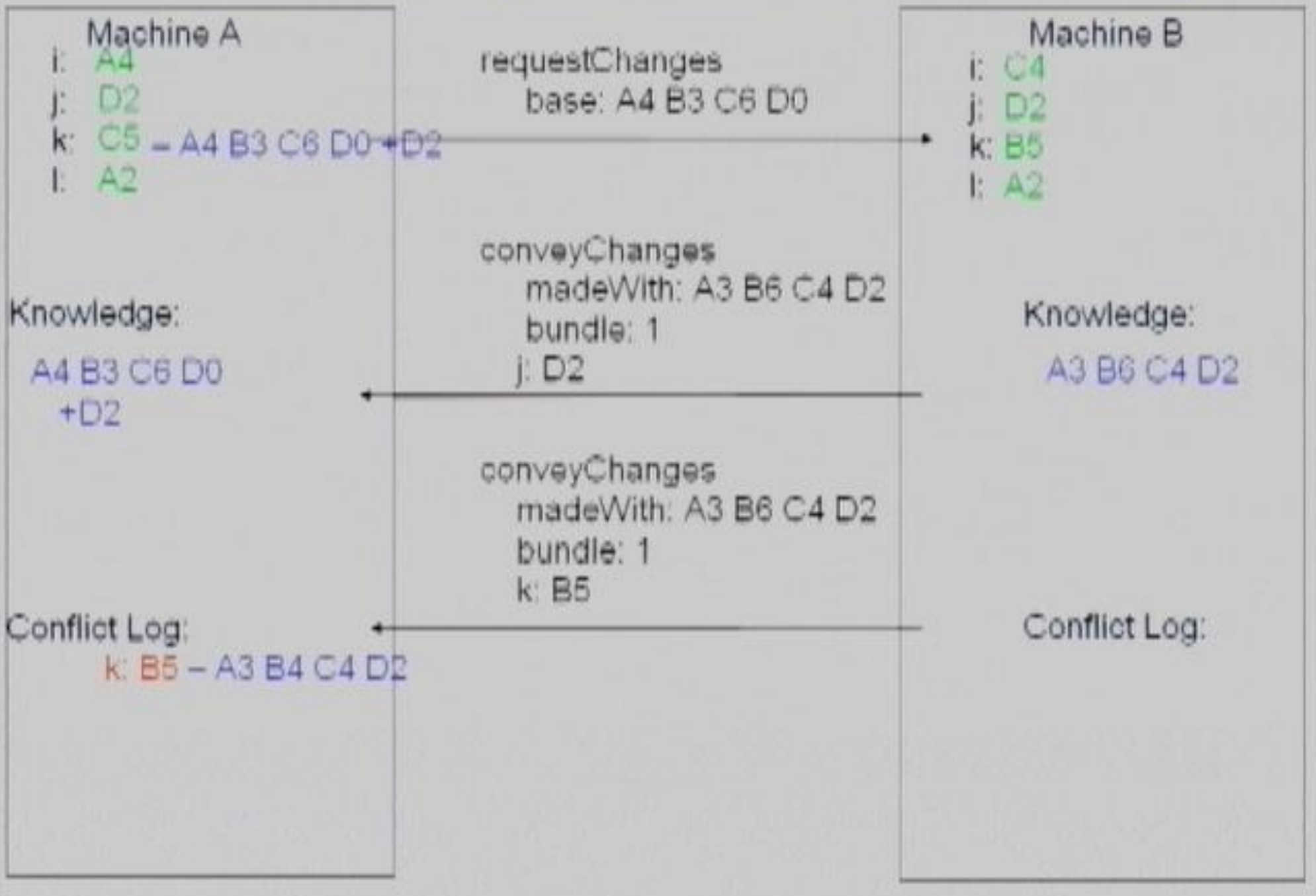
# Protocol with Implicit "Made With" Knowledge



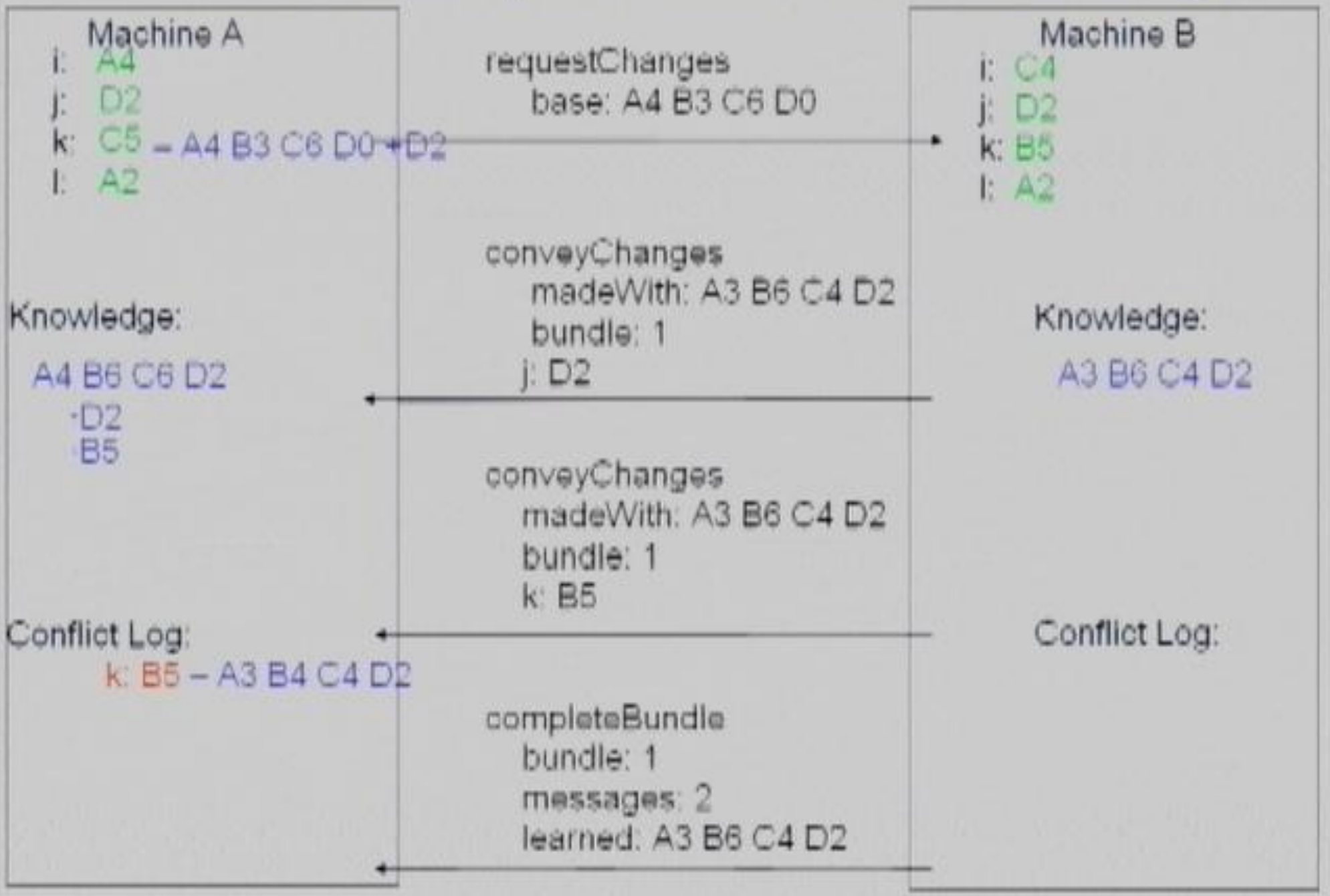
# Protocol with Implicit "Made With" Knowledge



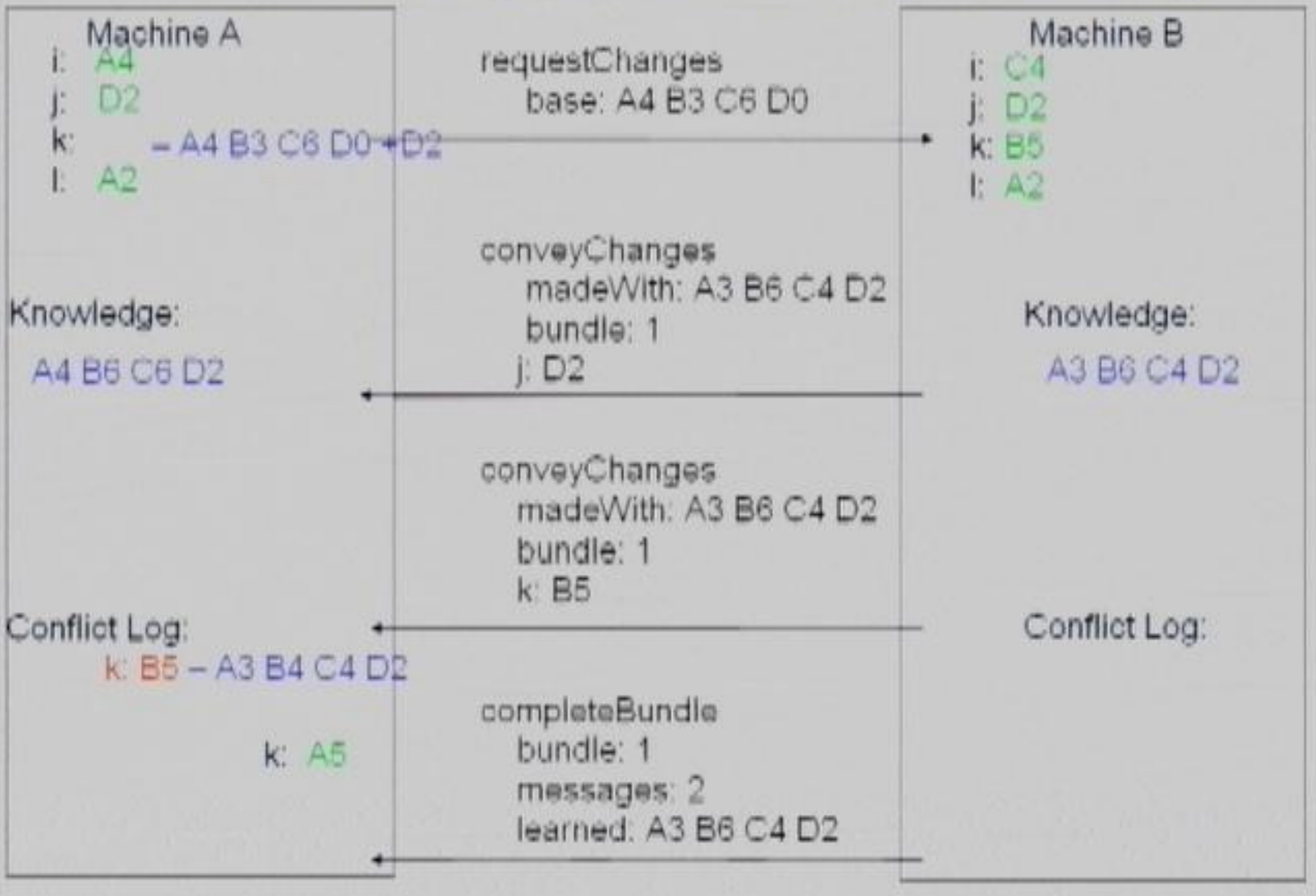
# Protocol with Implicit "Made With" Knowledge



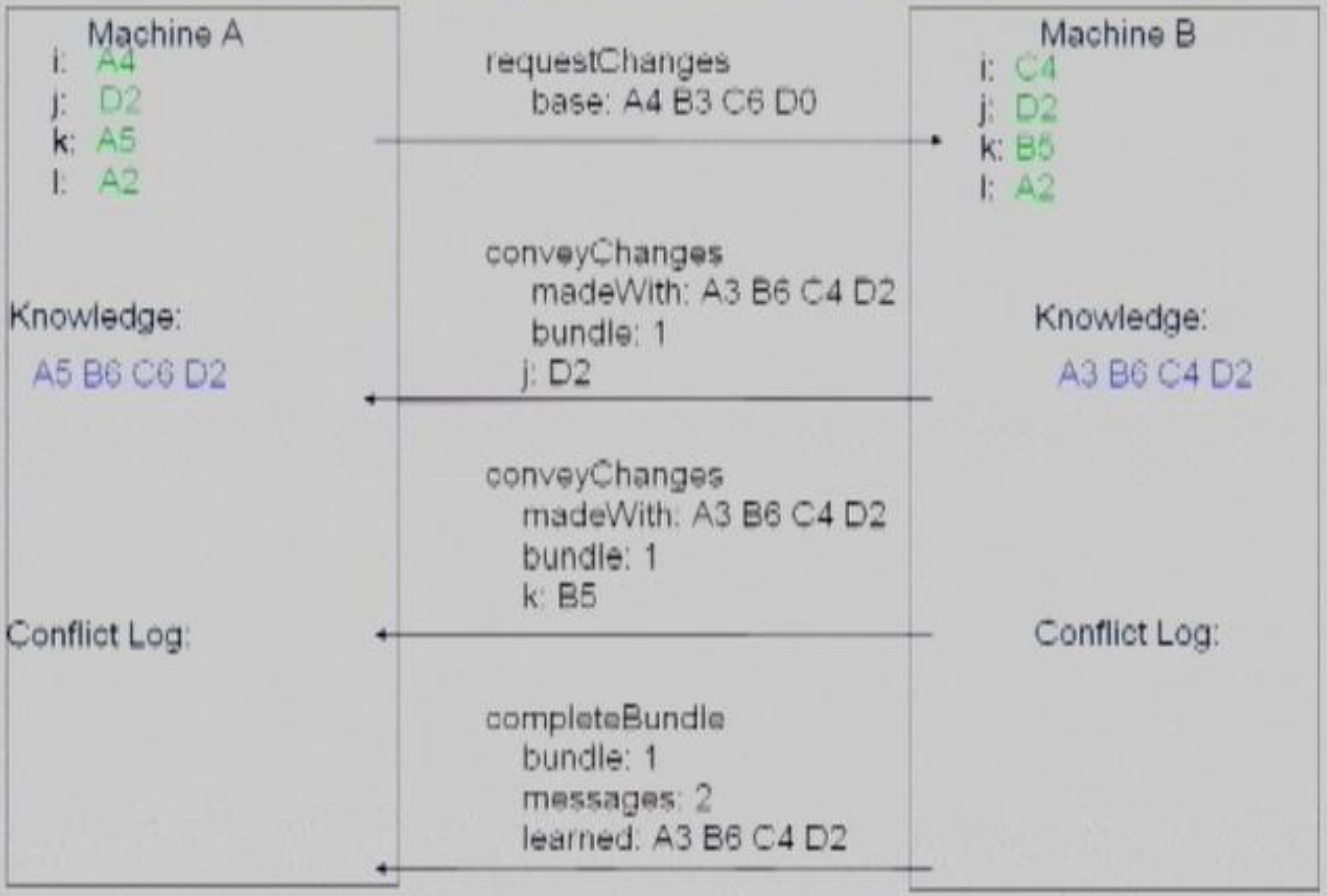
# Protocol with Implicit "Made With" Knowledge



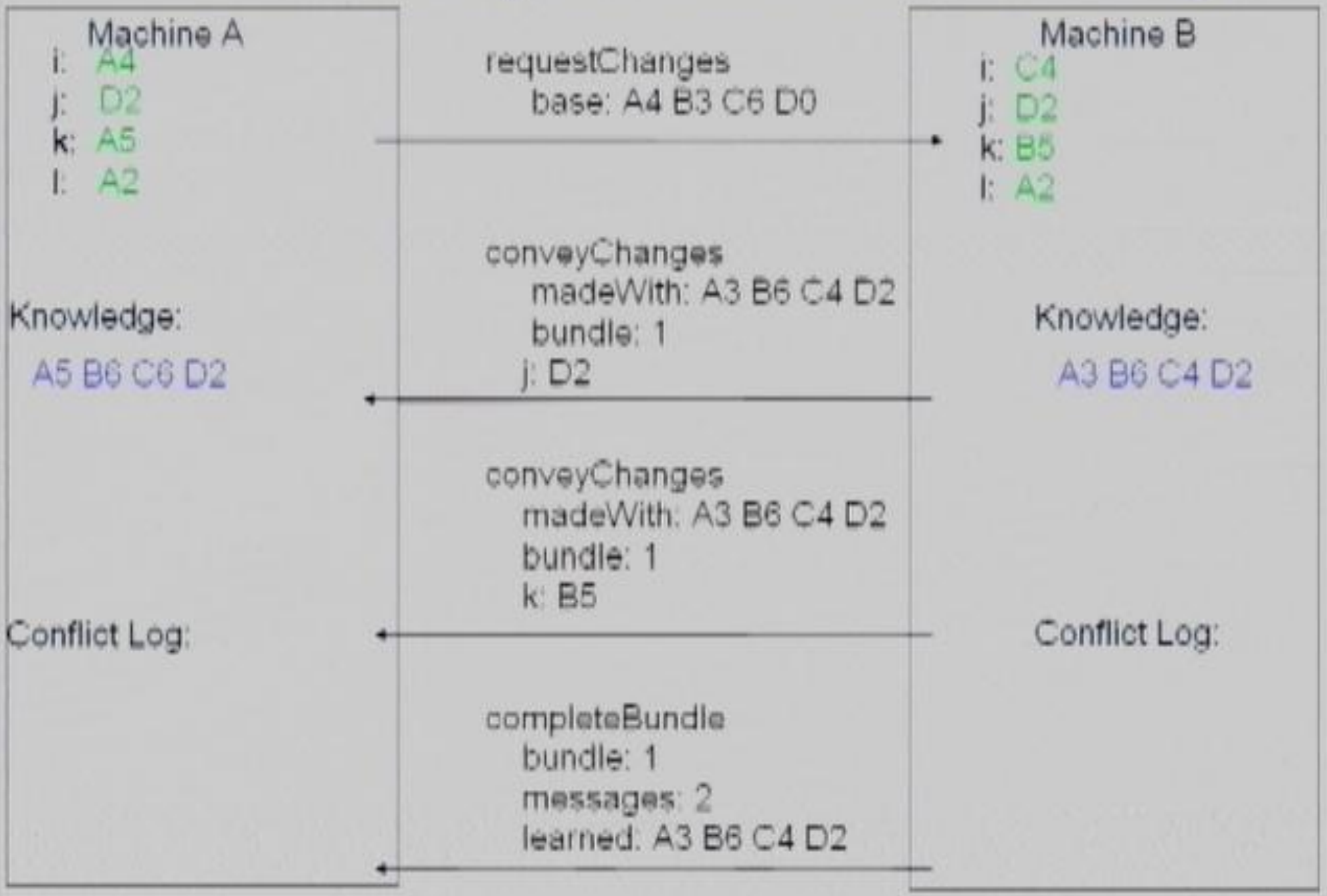
# Protocol with Implicit "Made With" Knowledge



# Protocol with Implicit "Made With" Knowledge



# Protocol with Implicit "Made With" Knowledge



# Summary

Key features of WinFS Synchronization:

- Favors availability over consistency
- Achieves robustness and scalability through pair-wise, knowledge-driven protocol
- Makes efficient use of bandwidth and storage
- Allows arbitrary communication topology
- Guarantees eventual convergence if well-connected topology
- Detects conflicts; permits manual or automatic resolution